

# X-ray imaging virtual online laboratory for engineering undergraduates

Corbi, Alberto; Burgos, Daniel; Vidal, Franck; Albiol, F; Albiol, A

**European Journal of Physics**

*DOI:*
10.1088/1361-6404/ab5011

Published: 02/12/2019

Peer reviewed version

# X-ray imaging virtual online laboratory for engineering undergraduates

**A Corbi[1], D Burgos[1], F Vidal[2], F Albiol[3] and A Albiol[4]**

[1]Research Institute for Innovation & Technology in Education (UNIR iTED), Universidad Internacional de La Rioja (UNIR), Spain, alberto.corbi@unir.net; [2]Bangor University, United Kingdom; [3]Instituto de Física Corpuscular (IFIC), Consejo Superior de Investigaciones Científicas (CSIC), Spain; [4]Department of Communications, Universitat Politècnica de València, Spain.

## Abstract

Distant-learning engineering students (as well as those in face-to-face settings) should acquire a basic background in radiation-matter interaction physics (usually in the firsts semesters). Some members of this category of scholars may feel some degree of aversion towards these types of pure sciences-related subjects (math, physics, chemistry, etc.). In online learning scenarios, the average student is already an adult (37 years old or above) and sees no special application of the aforementioned courses in his/her current or future professional life. Besides, online institutions tend to lean too much on applet-based simulations. These animated and interactive examples, although might shed some light on the theory associated to the studied physical processes, they also seem stripped down versions of the real events and are felt as disconnected from current scientific environments and engineering settings. For this reason, we describe a novel virtual lab approach to teach the basics of the low-energy interactions present in average X-ray settings. It combines real scientific simulation frameworks with modern computing techniques such as virtualization, cloud infrastructures, containers, networking and shared collaboration environments. It also fosters the use of hugely demanded development tools and programming languages and addresses the fundamentals of digital radiography and the linked electronic standards for image storage and transmission. With this mixed approach that blends scientific concepts, healthcare and state-of-the-art software solutions, our virtual labs have proven (over a period of 5 academic terms) to be very pedagogic and attractive (technically- and scientifically-wise) to online engineering undergraduates. For the sake of completeness, we also propose a hands-on activity that mimics the geometrical peculiarities of X-ray rooms with the help of visible light and cheap materials.

*Keywords*: X-ray physics, virtual laboratory, online learning, digital standards, collaborative environments, cloud technologies, containers.

Submitted to: *European Physics*

## 1 Introduction

First-year engineering students tend to see no point in going through the obligation of pursuing a *pure sciences*-related subject such as Physics. Its contents are often seen as *overly theoretical* and *unconnected to practice* [1]. Hence, in order to keep the students motivated, it is important to focus the teaching process of these subjects from a very practical and professional point of view and in a manner as much related as possible to daily engineering proceedings and standards. This need is even more *urgent* in online learning scenarios, where most undergraduates are already middle age adults (some of them even qualified experts in a given engineering field) seeking knowledge renewal or an official academic diploma [2].

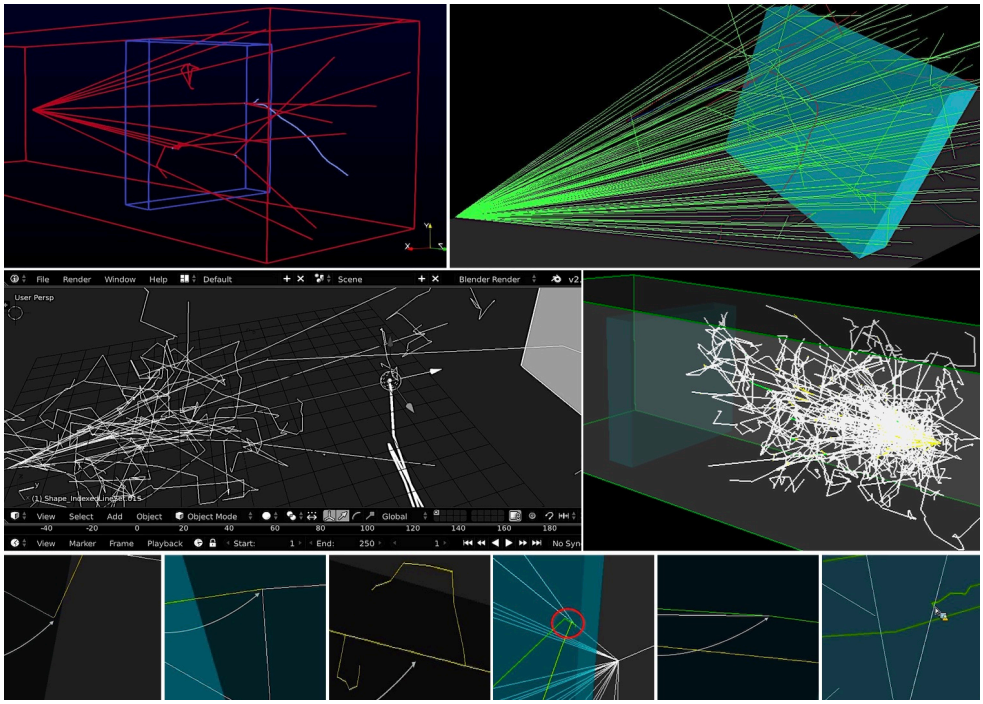

Figure 1: Results obtained with the proposed Geant4 simulation of X-ray photons and their interaction with matter (a virtual water-filled cuboid). These interactions (bottom row) comprise photoelectric and Compton effect, ionizations, pair production, bremsstrahlung, and even annihilation, respectively. The student can view all events in 3D, rotate the scene, alternate between solid/wireframe views, pan around and zoom-in at will. Many viewers are available: Blender, Paraview, Instant Player, view3dscene, etc. (discussed in Section 3.1).

Modern medical imaging is enabled by cutting edge engineering know-how (electronics, computing, software, materials science, Internet, etc.). Simultaneously, it is undoubtedly grounded on the understanding of basic physical processes: energy-matter interactions, radiation, atomic and particle physics, etc. Even the simplest radiological modality, such as primary diagnostic X-ray imaging, represents a relatively good scenario for learning key physical processes. Through the understanding of the underlying aspects of the radiographic production, a higher education engineering teacher can address the aforementioned issues related to the lack of motivation of his/her students. Besides, in contrast with computerized

tomography (CT), positron emission tomography (PET), or magnetic resonance (MR), this modality is undoubtably linked to everyday life (*who has not ever undergone an X-ray examination?*). It is also positively linked to preventive medicine and healthcare in general.

From our teaching experience, we have seen that through the clear comprehension of this *simple* radiological modality, an engineering student can feel he/she is understanding deep concepts about nature, and at the same time, learning productive (and transferable) technical expertise with tangible professional applications. Besides, the parent area of knowledge of conventional X-ray imaging and radiology, that is, medicine, is always accompanied by a large degree of social and health awareness, which unquestionably contributes to foster interest and funnel the learning experience. The technical and physical process around ordinary X-ray examinations can also be easily simulated with software-based platforms and with all levels of complexity, which turns out very appropriate for distant-education settings.

The main drawback in order to make use of medical imaging as a *unifying learning thread* in the engineering classroom is the difficulty (or even impossibility) of having access to the associated diagnostic equipment. Even conventional X-ray equipment, although present in almost any healthcare centre, is not accessible for research to plain students. This obstacle turns out most insurmountable in the case distant learning, where the classroom is now *delocalized*. As we stress in Section 6, these handicaps can be overcome with the help of modern simulation environments. Some of them are tackled in Section 3 for the specific case of conventional X-ray imaging. It is also possible to take further advantage of the situation by providing the future engineer with the necessary skills to master the latest technologies and industry standards. These have to do with: software manufacturing, development environments, programming languages, cloud infrastructures, information and network standards, and remote collaboration tools.

In this work, we summarize a real teaching experience that advances the exercise and acquisition of modern engineering competences, simultaneously with the learning of basic physical processes mostly related to electromagnetism and low-energy particle physics. This training is carried out in the form of virtual laboratories based on state-of-the-art online/offline computing tools. Additionally, the selected methodologies are easily deployable and platform independent. Almost no specific hardware or system is required as they are often built around standards and web (W3C) technologies. Although the experiences are aimed at higher education engineering studies, they can also be reimplemented with easier setups in pre-college scenarios where they can be managed by science teachers.

The paper is organized with a clear goal in mind: the description of the currently available tools and services for virtualizing the X-ray image generation practice. We also accomplish this objective by guiding the reader from the most technical and engineering-like point of view (in order to motivate the aforestated specific rage of students), to the easiest and most informative outlook. With more detail, in Section 2, we review previous works around the use of virtualization/simulation technologies to enable lab-like activities in STEM areas (for *Science*, *Technology*, *Engineering* and *Mathematics*). In Section 3, we remember the theoretical background behind X-ray diagnostic rooms and how it can be emulated through the help of virtual environments. In Section 4, we give an overview of the recent container technology approach and how it can be used in education to solve the problem of software/hardware incompatibility and platform dependence.

Given that we are targeting engineering students, we do not forget the pure technical side of the domain of radiology (information standards, networking protocols, image formats, etc.). We address this important aspect in Section 5. In Section 6, we transfer our solutions to *the cloud*, proving how it can be today used almost at no charge (thanks to free cloud services and/or free tiers) from mere and widely available web browsers. Modern collaboration and

scientific notebook solutions (that enables the concept of literate programming) are also discussed in the aforementioned section. In Section 7, we propose a simple (yet elegant) experiment that students can perform from home to mimic the geometrical behavior of X-ray equipment and radiography production. Finally, in Section 8, we show some results obtainable with the proposed virtual/remote lab approach and summarize our academic experience after having been implemented along several semesters in an online college institution. Section 9 summarizes our work and argues some conclusions.

## 2 Remote virtual labs in STEM

STEM subjects have the power of combining co-related subjects into one single *umbrella*. They share common interests and might use common resources. In distant education, remote virtual labs are a key part of almost any STEM subject. In other words, in online and blended settings, the use of remote virtual labs becomes the needed breakthrough to strengthen the liaison between the student, the teacher and the subject [3]. These labs facilitate a more than adequate layer to acquire the required competences [4, 5]. Even face-to-face settings also benefit from remote virtual labs since students and teachers can work at anytime and from anywhere without breaking the educational process [6, 7]. Further, thanks to digital academic protocols such as LTI (tackled in Section 6.4), administrative layers in the educational system are able to get precise activity reports on students and teachers (even if they are enjoying specific academic content provided by external actors). This fact contributes to developing a more accurate college management support that involves every single role in the system [8].

There are some commercial solutions in the market, like LabsLand†, that offer *pseudo hands-on* educational remote labs. They consist on real laboratories that can be controlled over the Internet through a computer, mobile device, or tablet, without having to install anything [9]. There is also Go-Lab‡, a European-funded project that works as a platform to author and share online labs into what they call *Inquiry Learning Spaces*. They provide controlled and restricted experiences on STEM that can be used and replicated by others.

Finally, some simple and affordable home-developed activities can be suggested to (and easily accomplished by) students in a *do-it-yourself* (DIY) scenario. For instance, in [10], the author discusses a very interesting experiment that imitates a PET modality with a homemade phantom and visible light. Although the experiment has some complexity, its remote execution can still be proposed to students as a final dissertation project. Besides, the activity has its *image reconstruction side* which entails the use of modern software tools, multi-image registration methods and attractive 3D visualizers. With the instructions detailed in [11], students are able to study the basics of signal processing with easy to use (and very affordable) USB dongles. In this research work, we also propose an activity related to the study of the geometry in X-ray settings that follows this hands-on/DIY spirit (Section 7).

## 3 Virtual X-ray environments

As commented in Section 1 and later deepen in Section 2, it would be a very difficult task to summon a group of students in an unoccupied X-ray room and offer them the possibility to experiment with real imaging equipment. This type of installations (deployed in hospitals and clinics) are usually in active service o worse, being serviced/tested/repaired. Besides, specific permissions are normally required and demanding regulations should be followed

† https://labsland.com
‡ https://www.golabz.eu

(i.e., radiological protection laws), not only for handling such apparatuses, but for being even near them while they are being operated. In the context of education and research, we cannot forget the need for phantoms (used by students in radiology practices) which are seldom very fragile and expensive. As previously highlighted, greater difficulties would arise in the case distant learning: collaboration agreements would be necessary between the online school/college and hospitals/clinics around the geographical scope of the academic action.

For this reason, a set of respected software packages has been carefully chosen to help the student have a first contact with the typical environment associated with the production of plain radiographic images ($\mathcal{X}$) and gain some insight of the physics beneath it. Each of the selected tools is highly used in its corresponding field (which contributes to deepen the *sense of usefulness* tackled in Section 1) and is presented to the engineering student in an attractive way, later discussed in Section 6. These computing resources not only bring physical reality to the learner through precise simulations, but also allow experimenting with processes and concepts otherwise impossible to *see* or *grasp* (e.g., the trajectory of particles, pair production situations, ionization events, etc.). The goal of these software packages is the (online) simulation of the basic physical (and also operational) processes that take part during a conventional X-ray examination, i.e., Compton/Rayleigh scattering, photoelectric effect and pair production (less probable). Outcomes are then presented in a very motivating way for engineering students, who are already used to electronic/computing standards/procedures.

As a quick summary, Compton scattered and Rayleigh X-rays contribute with no useful information to the formation of the final image. X-rays that undergo photoelectric effect do provide diagnostic information because of the fact of not reaching the detector. These *missing* X-rays are representative of anatomical X-ray-opaque structures. The photoelectric absorption of X-rays produces the light areas in a radiograph, such as those corresponding to the bones. Other X-rays pass through the body and reach the detector with no interaction whatsoever. They produce dark areas in a radiograph. The structures and tissues traversed are transparent to X-rays. An X-ray image results from the difference between those *photoelectrically absorbed* X-rays in the patient and those transmitted to the image receptor. This difference in X-ray interaction is called *differential absorption*. Approximately 1% of the incident X-rays reach the image receptor. Fewer than half of those interact to form an image. Thus, a radiograph is the result from just 0.5% of the X-rays emitted by the X-ray tube. In other words: *a radiograph can be compared with a shadow produced, not by visible light, but rather by Roentgen radiation when it traverses an object's internal parts*.

This concept of an X-ray image being comparable to a *shadowy bitmap* is very important and a special hands-on activity (independently doable by the students themselves at home) has been designed to address this topic (Section 7). In contrast with visible light, the X-ray beam is heavily attenuated as it interacts with matter. The attenuation mechanism takes place exponentially, following the well-known Beers-Lambert law, which can be very good (and fast) modelled with modern computing techniques, such as the ones described in Section 3.2 and Section 3.3. However, if we zoom-in into the physics underneath this *beam reduction process*, the student can also *see* and isolate by him/herself the important physics concepts reviewed above. In order to apply this *magnifying glass*, we have explored and implemented the framework detailed next in Section 3.1. The geometrical and optical implications of X-ray image production are also tackled in Section 3.2.3.

### *3.1 Simulation of X-ray interactions with tissues*

In order to carry out simple simulations of the well-know physical processes when undergoing an X-ray examination while simultaneously taking into account the pedagogical conditions

expressed in Section 1 and Section 2, we have chosen the Geant4 software package. Geant4 (*GEometry ANd Tracking*) is a toolkit for simulating the *passage* of particles through matter. It includes a complete range of functionality including tracking, geometry, physics models and collisions. It has been designed to handle complex geometries, and to enable an easy implementation through bindings and APIs with many computer languages. It is widely object-oriented and has been implemented in the C++ programming language. It has been used in applications in particle and nuclear physics, accelerator design, space engineering, education [12] and, of course, medical physics. Geant4 has also been used in cloud environments [13] with success (as we have also done in Section 6.1).

However, the default Geant4 C++ API (and this specific programming language in general) is somehow difficult to master [14] even among our target *academic audience* (middle age online college students) and this fact may mask the main learning goal of the medical physics lab presented in this work. Fortunately, Geant4 also has a healthy and more straightforward Python binding [15]. Python is a popular scripting language with an interactive interpreter [16]. Its raising reputation in engineering and science is reviewed with more depth in Section 3.2.5. Geant4Py is a Geant4-Python bridge (also known as G4Py†), which provides clean links towards many of the Geant4 classes. This enables the direct access to the Geant4 core from simple Python scripts. Perhaps, the most interesting class is `MedicalBeam`, which makes it straightforward to build, configure and *give life* to basic setups associated to the real-world clinical practice and radiological protection operations. In other words, thanks to this simple, yet elegant binding, it is possible to let the students play with this powerful framework with a few tens of lines of code. This very same setting has been previously implemented with pedagogic/academic success in the realm of education and medical application [17]. A basic code example is included in Listing 1. This code simply configures a medical mono-energetic beam of 20 low energy photons of 100 keV that collides with a virtual cuboid-shaped water phantom. The energy, geometry, composition of both the beam and the phantom are configurable and the student can change them at will while following the paradigm of *learning-by-doing* [18, 19]. The execution of this code outputs a nice, elegant and simple to render 3D view through which the student can freely navigate. With more detail, G4Py generates a VRML file (*Virtual Reality Modeling Language*) that can be viewed with many scientific (and real-world) software packages such as the acclaimed Paraview† visualization environment from Kitware. Paraview is open-source and binaries are available for most systems. Figure 1 shows some examples of the obtainable traces and some of the events that can be studied by students. By using this specific software package (Kitware's Paraview), engineering students can also learn the existence of other relevant software projects (including their home institutions, foundations and hosting companies) which are very central to the professional life of any engineer and scientist (CMake, ITK, VTK, etc., are a few examples). Another recommended 3D visualizer is Instant Player‡, which is also multi-platform and has been developed by the Fraunhofer Society. This simple fact (almost classifiable as *anecdotal*) also represents a good opportunity to introduce the existence of such reputed (but sometimes sadly unknown by engineers, both young and experienced) research institutions. A second open-source, multi-platform 3D visualizer is Blender§ (Figure 1), which although mainly targeted for animations and cinematic rendering, can also be used for scientific visualization [20]. As with Paraview, Blender is scriptable through the Python programming language. This fact enables the proposal of future, more

† https://gitlab.cern.ch/geant4/geant4/tree/master/environments/g4py
† http://www.paraview.org
‡ https://www.instantreality.org
§ http://blender.org

**Listing 1** G4Py code (also available at http://bit.ly/g4py_example) used to simulate a medical X-ray-like beam radiated towards a virtual water phantom. A screenshot of a live version of this code (hosted at MyBinder) is shown in Figure 7.

```
# Import necessary modules
import Geant4; from Geant4 import *; import g4py.NISTmaterials
import g4py.ezgeom; from g4py.ezgeom import G4EzVolume; import os.path
import g4py.EMSTDpl; import g4py.ParticleGun; import g4py.MedicalBeam
# Phantom & world description
phantom_width = 20; phantom_material = "G4_WATER"
g4py.NISTmaterials.Construct(); g4py.ezgeom.Construct()
g4py.EMSTDpl.Construct(); g4py.ParticleGun.Construct()
air = G4Material.GetMaterial("G4_AIR")
g4py.ezgeom.SetWorldMaterial(air); g4py.ezgeom.ResizeWorld(1*m,1*m,400*cm)
phantom_material = G4Material.GetMaterial(phantom_material)
phantom = G4EzVolume("PhantomBox")
phantom_zwidth = phantom_width*cm; phantom_zlocation = 100.*cm
phantom.CreateBoxVolume(phantom_material,100*cm,100*cm,phantom_zwidth)
phantom.SetColor(0,0.9,1.0)
phantom_box_pv = phantom.PlaceIt(G4ThreeVector(0*cm,0*cm,phantom_zlocation))
# Particle beam description
particle_type = "gamma"; num_particles = 20; particle_energy = .1;
beam = g4py.MedicalBeam.Construct(); beam.particle = particle_type
beam.kineticE = particle_energy*MeV;
beam.sourcePosition = G4ThreeVector(0*cm,0*cm,-90*cm)
beam.fieldXY = [120*cm,120*cm]; beam.SSD = 190*cm
# Visualization commands
gApplyUICommand("/run/initialize"); gApplyUICommand("/vis/viewer/flush")
gApplyUICommand("/vis/open VRML2FILE")
gApplyUICommand("/vis/viewer/refresh")
gApplyUICommand("/vis/scene/create")
gApplyUICommand("/vis/scene/add/volume");
gApplyUICommand("/vis/drawVolume")
gApplyUICommand("/vis/sceneHandler/attach")
gApplyUICommand("/tracking/storeTrajectory 1")
gApplyUICommand("/vis/scene/add/trajectories")
gApplyUICommand("/vis/scene/add/hits")
gApplyUICommand("/vis/scene/endOfEventAction accumulate")
# Launch simulation
rand_engine = Ranlux64Engine()
HepRandom.setTheEngine(rand_engine); HepRandom.setTheSeed(20050830L)
gRunManager.Initialize()
gRunManager.BeamOn(num_particles)
```

complex and more interesting activities for the students and the fostering the *artistic side* of this virtual lab. For this reason, we believe this activity, initially classifiable as STEM (concept tackled in Section 1), can *de facto* be turned into a STEAM one (where the *A* stands for *Arts*). A third open-source, multi-platform visualizer is view3dscene†, which is the foundation of a free open-source 3D/2D game engine using modern Object Pascal. Finally, modern web standards (i.e., WebGL) also allow the representation of the outcome of state-of-the-art particle simulations [21]. An example of this feature is shown in Figure 7-left and a ready-to-play HTML file can be downloaded from http://bit.ly/g4py_webgl. Anyway, independently of the 3D visualizer and in the context of the here proposed medical physics remote lab, students are simply requested to carefully identify and visually highlight

† https://castle-engine.io/view3dscene.php

all the possible events and cite their theoretical background.

Even though the G4Py is very easy to use, its implementation (together with a complete Geant4 installation) it might still be felt as some difficult to achieve by first-semester students in their own home computers (we recall this work is mainly focused at distant education scenarios). There are no Geant4+G4Py installers available for all the necessary platforms and OS types and versions. This is the reason why we have devised a novel container-based solution explained in Section 6 and discussed the use of cloud infrastructures in Section 6. Nevertheless, the compilation and deployment of Geant4 and G4Py from their respective source code repositories can turn out motivating for students with an engineering profile.

### *3.2 Virtual radiographs from simple geometrical shapes*

In the simulation carried out in Section 3.1, the student has learnt, played, and almost *visually* experimented with the underlying physical processes in X-ray examinations. Nevertheless, he/she has not had yet any contact with the concept of *X-ray image*. The reason for this is that we have not yet introduced the notion of *detector*.

The next logical step after simulating a basic medical low-energy electromagnetic beam and letting it interact with an elemental object (homogeneously composed of a given substance) is the derivation of simple radiographs from solid geometrical shapes. These shapes can be described as STL or PLY [22] data structures. Following the magnifying glass metaphor introduced at the beginning of this section, we can now *zoom out* a little bit from the realm of the most basic photon-matter interactions and redirect our pedagogical efforts towards the *graphical side* of the X-ray image production, which is mainly influenced by the attenuation process. At this zoom-level, the student also begins to gain some insight about the optical aspect of the generation of radiographs. Here, a radiograph from a homogeneously composed tessellated object (e.g., a virtual triangle-mesh *filled* with a single substance) will only depend on the so-called *length's buffer*. The geometry of the X-ray setting is also important, as will be tackled in Section 3.3.

The $L$-buffer is a *mask of the bisections* (inside the radiographed object) by those rays of the original X-ray beam that reach the detector. This 2D matrix is linked to a specific radiograph and examination. For each pixel $p(p_x, p_y)$ in the *imaging plate*, we can attach a length $l(p)$, as depicted in Figure 2. In order to compute the length of a ray through a person's body, we use the *photon transport* algorithm Section 3.2.1 and Section 3.2.2 and devised by one of the coauthors of this paper. This method makes extensive use of the GPU and can compute the set of $l(p)$ very fast. The $L$-buffer can be translated to a grayscale image $\mathcal{L}$ linked to each radiograph. The intensity of each pixel stands for the distance covered by a ray within the body. The $L$-buffer can also be represented as an extra $z$ dimension connected to the original X-ray image (adding new geometrical data to the radiological information.

*3.2.1 gVirtualXRay* is mainly focused on simulating X-ray images on the graphics processor unit (GPU) using OpenGL. The library is written in C++ and the OpenGL Shading Language (GLSL). It is portable and works on a wide range of computers and operating systems. Wrappers to other popular languages, including Python, R, Ruby, and GNU Octave, are also provided. The source code is available on SourceForge† under the BSD 3 license.

The use of GPU computing opens up new perspectives. A high resolution X-ray projection can be simulated in a few microseconds, rather than weeks with MC methods, which makes it possible for students to visually and interactively observe the influence of

† https://sourceforge.net/projects/gvirtualxray

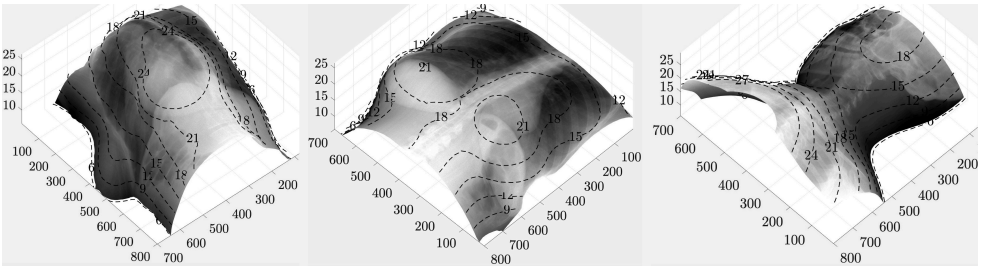

Figure 2: Examples of $L$-buffers. They are represented as 3D shapes whose height is the traversed span within the patient (woman X-rayed in oblique, frontal and lateral projections).

each parameter of the imaging system. gVirtualXRay can also be used in: interactive medical simulations for training purposes [23, 24, 25], simulation of realistic data acquisitions in medical CT with patient respiration [26], 3D volume to 3D meshes registration [27], and the determination of artifacts in CT for material science applications [28].

*3.2.2 Deterministic simulation and the Beer-Lambert law* provides a good compromise between speed and accuracy [28]. The Beer-Lambert law models the attenuation of photons based on the properties of the material through which they are traveling:

$$N_{out}(E) = N_{in}(E) \times e^{(-\int \mu(E,\rho(x),Z(x))\cdot \mathrm{d}x)} \tag{1}$$

where $N_{in}(E)$ is the amount of photons at energy $E$, $N_{out}(E)$ is the amount of transmitted photons and $\mu$ is the linear attenuation. $\mu$ depends on: $E$ (energy of initial photon stream), $\rho$ (density of the object), and $Z$ (chemical element of the material). When the composition of the scanned objects is not homogeneous, Eq. (1) can be expressed as a discrete assemble:

$$N_{out}(x,y) = N_{in} \times \exp\left(-\sum_{i=0}^{i<objs} \mu(i,E)\cdot L_p(i,x,y)\right) \tag{2}$$

where $N_{out}(x,y)$ is the number of transmitted photons received by the detector at the pixel $(x,y)$, $objs$ is the total number of radiographed objects, $l(i,x,y)$ is the path length of the ray in the $i^{\text{th}}$ object, and $\mu(i,E)$ is the $\mu$ at energy $E$ for the associated object. The 2D image of $l(p)$ values for a given object is called $L$-buffer. It aims at computing the path length of X-rays through polygon meshes, from the X-ray source to every pixel of the detector. In practice, the total energy received by the detector at every pixel is recorded as:

$$E_{out}(x,y) = N_{out}(x,y) \times E \tag{3}$$

*3.2.3 Geometry in virtual X-ray simulation frameworks* is schematically represented and explained in Figure 4. The particularities of the chosen geometry represent a good opportunity to emphasize, among engineering students, some key aspects of treating X-ray equipment as plain *pinhole* cameras. For instance, in X-ray cameras, *the focal length is not constant*. The principal point also changes from examination to examination, which is in contrast with traditional pinhole apparatuses. This fact allows the student to also reflect on some collateral concepts related to optics and image formation (which is a key subject in many technical studies that involve sensing equipment and signal detection).

X-ray devices are in fact composed of a Röntgen radiation source and a *disengaged* sensitive surface: the *flat panel detector* (FPD) or *imaging plate* (IP). One of the key

difference is that, in the case of X-rays, a projected point $\mathbf{Q}_i$ is situated *between* the anode $C$ (optical center) and the FPD. In other words, every $\mathbf{Q}_i$ point is projected to a 2D mark in a defined coordinate $\mathbf{q}_i$ in the detector. $C$ is also the starting point of the photon stream, whereas in a plain camera, the light source is the *recorded scene* itself, which emits *echoed* light. This dispersed light *enters the camera* through the pinhole and finally *collides with* the FPD. A slide projector is the source of the light stream as well. However, its focal length is fixed (we only get a sharp image at a specific distance) and it is possible to derive it [29].

As already stated, in Section 7, we propose a neat manual activity that allows students to assimilate these concepts and properties with the help of plain light sources, casted shadows, simple cameras and a bit of software-mediated matrix calculus. The geometry of an X-ray room is also a very important feature to be taken into account because it is intimately connected to the process of radiograph formation. With the tools already discussed in Section 3.1 and Section 3.2.4, the student can recreate, compose and intuitively visualize by him/herself the radiographic scene (Figure 11).

*3.2.4 Implementation details* of gVirtualXRay [30] are also key if we want the student to interact with this framework. It is very portable and works on a wide range of systems. Wrappers to other popular languages, including C#, Java, GNU Octave, Perl, Python, R, Ruby and Tcl, are also provided. The source code is available under the BSD 3 license.

The gVirtualXRay framework implements Freud *et al*.'s $L$-buffer principle [31] on the GPU. It relies on a modified rendering pipeline from 3D computer graphics. Equations Eq. (2) and Eq. (3) can be implemented as a *rasterization* process using the programmable graphics pipeline of the GPU (see Figure 3) for a further increase of speed [32]. Rasterization is a real-time 3D rendering technique that creates a 2D image of pixels from a 3D scene. They are implemented using multi-pass rendering. Each rendering pass corresponds to a rasterization phase and the result of the previous rendering passes are used to generate the final image. In gVirtualXRay, the scanned objects are defined by their 3D primitives (here triangles made of 3 vertices expressed in 3D modelling coordinates) and their material properties. A material can be identified by a chemical element, a compound or a mixture. To generate the $L$-buffer of an object, blending (transparency) is enabled. If needed, modelling transformations can be used to scale, rotate and/or translate the geometry of the object being processed.

At this stage, its geometry is modelled in 3D world coordinates. The transform is applied so that the shifted/rotated object is viewed from the X-ray source and aligned relative to the centre of the detector. Its geometry is now in 3D camera coordinates. Its triangles are projected onto the plane corresponding to the detector. This transformation depends on the X-ray source position, detector centre, resolution and orientation. Its geometry is now in 2D screen coordinates. The primitives that are not in the viewing space between the source and detector are discarded in the clipping stage. The resulting image is computed offline into a texture using a *framebuffer object* [33] from the OpenGL library [34]. Figure 3 shows the graphics pipeline to compute the $L$-buffer. Once it is computed for all objects, a new rendering pass is executed using blending to efficiently derive the sum ($\sum$ sign) in Eq. (2).

This approach allows gVirtualXRay to provide simulated X-ray projections in real-time without any further compromise on accuracy [35]. For more realism, it is also possible to take into account the imaging acquisition chain parameters, e.g. finite size of the X-ray tube, spectrum of the radiation beam, and the X-ray detector's impulse response [35, 36]. In a polychromatic case, with $M$ different energies in the incident beam spectrum, it becomes:

$$E_{out}(x,y) = \sum_{j=0}^{j<M} E_j \times N_{in}(E_j) \times \exp\left(-\sum_{i=0}^{i<objs} \mu(i,E_j)\cdot L_p(i,x,y)\right) \quad (4)$$

where $E_{out}(x, y)$ is the total energy at the detector and at the pixel $(x, y)$. When the source is defined by a set of points instead of a single small point, it becomes:

$$E_{out}(x,y) = \sum_{k=0}^{k<P} \sum_{j=0}^{j<M} E_j \times N_{in}(E_j) \times \exp\left( - \sum_{i=0}^{i<objs} \mu(i, E_j) \cdot L_p(i, k, x, y) \right) \quad (5)$$

where $P$ is the amount of points defining the contour of the X-ray source and taking into consideration changing point sources ($k$) in $L_p(i, k, x, y)$. Again, each sum in Eq. (5) corresponds to a rendering pass that is executed by means of a fast blending stage.

*3.2.5 Student implementation* should be easy and platform independent. In the context of the present research work and as also happened with the Geant4 C++ API implementation, the C++ API of gVirtualXRay might be felt as *aversive* by some first semester college students. Fortunately, as with Geant, gVirtualXRay also includes modern bridges for other *more lightweight* computer languages, such as Python, Ruby or Octave [37].

For this purpose, gVirtualXRay makes use of the SWIG compiler [38]. SWIG links code written in C and C++ with scripting languages (Perl, Python, Ruby, Tcl, etc.). It takes the declarations found in C/C++ header files and generates the associated code that interpreted languages need to access the underlying C/C++ scaffold. Listing 2 and Listing 3 show a small code example of the application of the Ruby and C language bindings, respectively. This architectural bridges represent a good opportunity to introduce the young first semester engineer to the aforementioned, massively used, scripting technologies. However, this SWIG-based setup relies on the compilation and correct deployment of the gVirtualXRay library for each system, which might turn out a complex process for some students (although a nice installation guide can be reached at http://bit.ly/gvxr_guide and it might turn out a motivating computing exercise for engineering students, as discussed at the end of Section 3.1). For this reason, we suggest a container-based solution presented in Section 4.

For the sake of completeness, the author of gVirtualXRay offers some pre-compiled GUI applications to play with the simulations in several systems (Figure 3). Although this small user-land applets may turn out useful to gain some insight about the internals of this

**Listing 2** Ruby code for performing the same simulation of a radiograph from a tessellated object with gVirtualXRay. The code is available at http://bit.ly/gvrx_ruby.

```
require 'gvxrRuby'
require 'matplotlib'
Matplotlib.use("TkAgg")
require 'matplotlib/pyplot'
plt = Matplotlib::Pyplot
GvxrRuby.createWindow()
GvxrRuby.setWindowSize(512, 512)
GvxrRuby.setSourcePosition(-40.0,
    0.0, 0.0, "cm")
GvxrRuby.usePointSource()
GvxrRuby.setMonoChromatic(0.08, "MeV"
    , 1000);
GvxrRuby.setDetectorPosition(10.0,
    0.0, 0.0, "cm")
GvxrRuby.setDetectorUpVector(0,0, -1)
GvxrRuby.setDetectorNumberOfPixels
    (640, 320)
GvxrRuby.setDetectorPixelSize(0.5,
    0.5, "mm")
GvxrRuby.loadSceneGraph("dragon.stl",
     "mm")
for i in 0..(GvxrRuby.
    getNumberOfChildren("root")-1) do
 label = GvxrRuby.getChildLabel('root
     ', i) GvxrRuby.moveToCentre(
     label)
 GvxrRuby.setHU(label, 1000)
end
GvxrRuby.disableArtefactFiltering()
x_ray_image = GvxrRuby.
    computeXRayImage()
plt.imshow(x_ray_image, cmap="gray")
plt.savefig('xray ruby.png')
GvxrRuby.displayScene()
```

**Listing 3** C code for performing the same simulation of a radiograph from a tessellated object with gVirtualXRay. The code is available at http://bit.ly/gvxr_c.

```
#include <cstdlib>
#include <string>
#include <iostream>
#include <exception>
#include "SimpleGVXR.h"
int main(){
// Create an OpenGL context
 createWindow();
 setWindowSize(512, 512);
// Position of the X-ray source
 setSourcePosition(-40, 0, 0, "cm");
// Shape of the X-ray source
 usePointSource();
// Spectrum of the X-ray beam
 setMonoChromatic(0.08, "MeV", 1000);
// Position of the detector
 setDetectorPosition(10, 0, 0,"cm");
// Orientation of the detector
 setDetectorUpVector(0, 0, -1);
// Number of pixels of the detector
 setDetectorNumberOfPixels(640, 320);
// Distance between two pixels
 setDetectorPixelSize(.5, .5, "mm");
 // Load a polygon mesh from STL file
 loadSceneGraph("dragon.stl", "mm");
// Material properties
 int c = getNumberOfChildren("root");
 for (int i = 0; i < c; ++i){
  std::string label =
    getChildLabel("root", i);
  moveToCentre(label);
  setHU(label, 1000);
 }
 computeXRayImage(); // Compute image
// Save the image into a image file
 saveLastXRayImage("dragon.tif");
 return EXIT_SUCCESS;
}
```

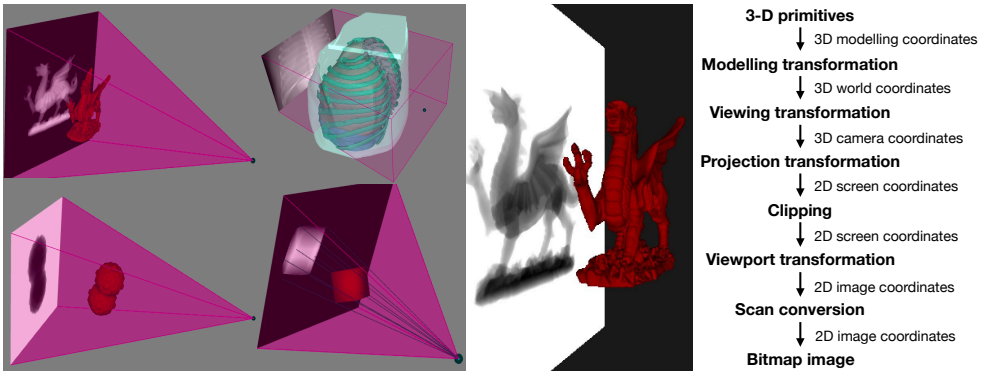


Figure 3: Left: Examples of the GPU-based apps offered by gVirtualXRay. These tools can render each frame in a few milliseconds (fostering the *sensation of real-time* while the student adjusts the associated geometrical parameters). Right: Operational pipeline of gVirtualXRay.

simulation framework, we believe it is also important (from an engineering point of view) to play with its low-level API and internals. These apps are also very pragmatic to quickly grow the student's intuition about the geometry in X-ray settings (tackled in Section 3.2.3), which is often overlooked even by ordinary medical physics M.Sc. studies.

### *3.3 Digitally reconstructed radiographs from volume-density*

A next phase in our lab is the derivation of radiographs from volumes with multiple densities. These volumes have been previously isolated from CT data. In these structures, each voxel contains a number acquainting for the density of that 3D point in Hounsfield units. In Section 6.4 we will show how students can generate these volumes by themselves with simple

tools. However, to begin with, the teacher can initially deliver these *pre-composed* files.

The file format most used for storing this kind of information (used even beyond the radiological sphere) is MetaImage, devised by the Kitware's ITK [39] project. A volume consists in voxels that are organized in a 3D array that contains sampled data. In a CT volume, these voxels contain information in Hounsfield units (HU). Although not much present in the realm of primary care X-ray imaging, the HU is also a key concept in medical physics that any future engineer should, at least, know about. The HU scale is a linear transformation of the original linear attenuation coefficient ($\mu$). This mathematical relation is given by the relation $HU = 1000 \times (\mu - \mu_{\text{water}}/\mu_{\text{water}} - \mu_{\text{air}})$, where $\mu_{\text{water}}$ and $\mu_{\text{air}}$ are the linear attenuation coefficients of water and air, respectively. Despite the HU scale being reserved for more complex modalities such as CT, the concept of radiation attenuation is extremely important and should be tackled by engineering students no matter their area of specialization.

Generating a realistic radiographic image from such volumes is equivalent to obtaining a DDR or *digitally reconstructed radiograph*. A DDR is a derived radiograph which can be generated from a CT scan. It can provide valuable diagnostic information by themselves and can be, for some type of assessments, as useful as plain radiographs and a diagnosis complement to the CT studies they are derived from. Many vendors ship and/or provide DRR software and tools. For our medical physics lab, we have chosen the Plastimatch package [40], which is mainly developed at Harvard University. Plastimatch [41, 42] implements a fork of the Siddon ray tracing method [43]. This algorithm uses the original exact path length based on the intersection of rays with the image voxels. From here, voxel interpolation is also applied which contributes to increase the apparent resolution of the final DRR. Both multi-core and GPU versions are available. The geometrical framework defined by Plastimatch is also shown in Figure 4, which is very similar to that used by gVirtualXRay. Plastimatch also implements a similar photon transport algorithm to that explained in Section 3.2.1. However, the *incremental Siddon algorithm* [44], utilized by Plastimatch computes the attenuation values from a CT dataset and then performs a ray tracing phase from the source to a detector plane. The line integral of the attenuation coefficients along the ray is calculated. The process is then iterated for different rays until the final image is achieved. The algorithm needs some input values such as kVp, voxel size and even the angle of projection as input parameters.

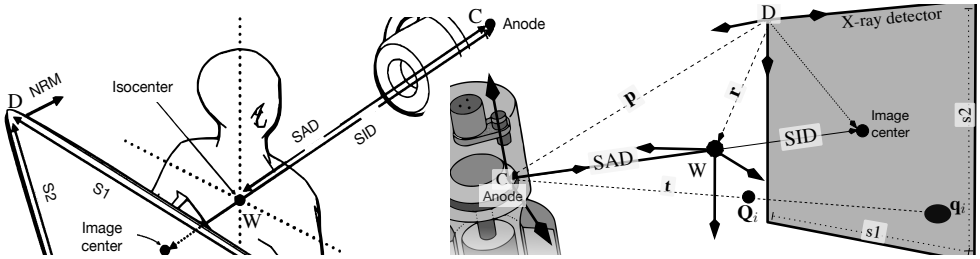


Figure 4: Two views of the geometry in a simulated X-ray setting. The SAD is the distance from the anode (C) to a fixed point in space, also known as *isocenter* (W). In the computing frameworks used in this work, W is typically pre-established by the coordinate system of the PLY/STL mesh or the MetaImage volume used in the simulation. The SID is the distance from C to the center of the virtual detector and it is equivalent to the focal length of the X-ray system. D is a coordinate system whose origin is at one of the corners of the detector and is equivalent to the `-nrm` option in the Plastimatch DRR generation tool. The parameters s1 and s2 are the width and height of the detector, respectively. The right image also shows how a given 3D point $\mathbf{Q}_i$ is project in the detector as $\mathbf{q}_i$.

Plastimatch offers Windows binaries by default, however, they can be run on non-Windows systems (Linux, macOS, etc.) thanks to the Wine project [45]. This fact represents a humble opportunity to introduce this important and robust software project to engineering students that allows the execution of Windows binaries on top of Unix-like environments.

For instance, once Plastimatch is installed, it is possible to quickly obtain a simple DRR form a volume with the following command:

```
drr –nrm "–1 0 0" –g "1000 1080" –r "200 200" –z "260 260" –o \
"100 105 125" isabelix.mha; convert out_0000.pfm isabelix.png
```

where the options are: `-nrm x y z` to set the normal vector for the panel, `-g sad sid` to set the SAD/SID values (in mm), `-r r c` to define the output resolution (in pixels), and `-z s1 s2` to set the physical size of detector (also in mm). The `convert` command transforms the image to a more widely used 16-bit image format. This command is provided by the ImageMagick package [46], which offers ready-to-use binaries for most platforms.

Nevertheless, in spite of the apparent availability of Plastimatch (to be run in almost any system), we believe future engineers should use more state-of-the-art environments and tools like the ones discussed in Section 4

## 4 Complex simulations and modern virtual containers

As stated in Section 3.1 and Section 3.2, the problem with complex (and open-source) simulation frameworks is that they need to be compiled, configured and deployed in the wide range of computer systems. This range can turn out huge and overwhelmingly variated in the case of online learning, where enrollment rates can grow up to the hundreds and thousands of students. This is the case of MOOCs (*Massive Open Online Courses*). For this reason, we now emphasize the exploitation of modern virtual environments, such as virtual containers.

Containers rely on virtual isolation to run applications that access a shared operating system (OS) kernel. Container images include the necessary information to run a specific application/process via a *container engine*. Applications that are *containerized* can be composed of several images that fit together like puzzle pieces. Containers do not retain session information. Multiple instances of an image can run at the same time, and new instances can smoothly replace crashed ones. In the context of this work and as it can be seen in Figure 5, containers have been used to smoothly work with the simulation frameworks presented in Section 3.1 and Section 3.2. As affirmed above, containers hold the *sine qua non* components to run almost any desired software. These components comprise files,

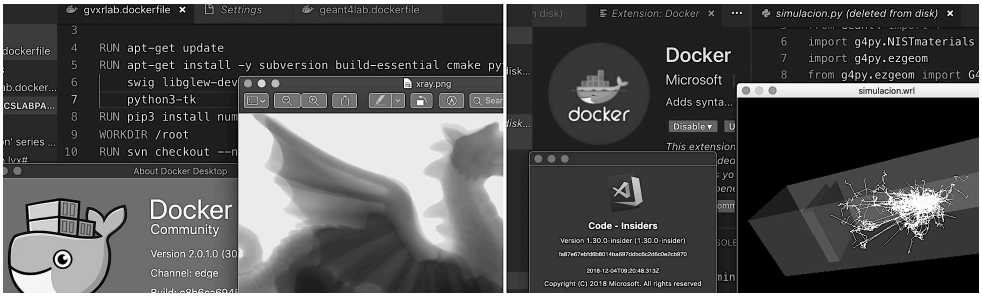


Figure 5: Examples of containerized simulations showing the output of gVirtualXRay (left image) and Geant4 (right image), respectively.

environment variables, and frameworks. The host OS limits the container's access to the physical resources, such as the CPU, GPU, storage and memory. Container image files are executable versions of applications or services and differ from one implementation to another.

The most famous concretization of the container technology is Docker†. This specific implementation has already been proposed as a perfect tool for both education [47, 48] and science [49]. Docker images are made up of multiple layers, which start with a base image [50]. Each image contains a writable layer on top of static layers.

Docker uses a configuration language (known as *Dockerfile*) to control the definition of an application container (files to be included, networking status, processes to be run, etc.). For instance, Listing 4 shows the Dockerfile needed to build (and run) an image configured with the gVirtualXRay framework (whose technical implementation has been discussed in Section 3.2.5) and the required Ruby, Python and GNU Octave bindings.

**Listing 4** Example of a Dockerfile (also available at `http://bit.ly/g4pylab_dockerfile`) that generates a virtual Docker image with the gVirtualXRay framework preinstalled and ready to work with its Python, Ruby and GNU Octave bindings. The last line (`CMD […]`) specifies the command (or set of commands) to be primarily run by the container. In our case, these instructions consist in the simulation scripts listed in Listing 2 and Listing 3.

```
FROM ubuntu:18.04
RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -y
    subversion build-essential cmake python3-dev wget zlib1g-dev ruby-dev
    swig libglew-dev xorg-dev libx11-dev xorg-dev fftw3-dev python3-pip
    libassimp-dev python3-tk xvfb octave-pkg-dev octave-image libglfw3-dev
RUN apt-get install --fix-missing
RUN pip3 install numpy matplotlib image pillow
ENV PYTHON /usr/bin/python3
RUN gem install matplotlib
WORKDIR /root
ADD gvxrsource.tgz /root/
RUN ls /root/gvirtualxray-trunk
RUN mkdir GVXRbuild
RUN mkdir GVXR
COPY XCOM /root/GVXR/XCOM
WORKDIR /root/GVXRbuild
RUN cmake ../gvirtualxray-trunk -DBUILD_PYTHON3=ON -DBUILD_RUBY=ON \
 -DUSE_SYSTEM_XCOM=ON -DXCOM_PATH=/root/GVXR/XCOM -DUSE_SYSTEM_ASSIMP=ON \
 -DBUILD_OCTAVE=ON -DUSE_SYSTEM_GLFW=OFF
RUN make -j2
WORKDIR /root
RUN echo "addpath('/root/GVXR')" > /root/.octaverc
RUN cp -R /root/GVXRbuild/tools_bin/Wrappers/python3/* /root/GVXR/
RUN cp -R /root/GVXRbuild/tools_bin/Wrappers/ruby/* /root/GVXR/
RUN cp -R /root/GVXRbuild/tools_bin/Wrappers/octave/* /root/GVXR/
RUN rm -rf GVXRbuild gvirtualxray-trunk
WORKDIR /root/GVXRLab
ENV PYTHONPATH $PYTHONPATH:/root/GVXR
ENV RUBYLIB $RUBYLIB:/root/GVXR
CMD ["sh","-c","/usr/bin/xvfb-run -a /usr/bin/python3 simulacion.py ; /usr/
    bin/xvfb-run -a /usr/bin/ruby simulacion.rb ; /usr/bin/xvfb-run -a /usr/
    bin/octave simulacion.m"]
```

Containers can be effortlessly distributed through public or private repositories. Docker itself hosts an open public repository of containers, called the *Docker Cloud* or *Docker Hub*.

† `https://www.docker.com`

Some of the images presented in this work are openly (and freely) hosted there†. A student would only need to run the following command in his/her system:

```
docker run pammacdotnet/geant4lab –v local_folder:/root/GVXR
```

where `local_folder` is a local shared directory in the student's physical computer. This directory hosts the scripts listed in Listing 2 and Listing 3 (and are therefore, editable) and receives the generated virtual radiographs when the container is run.

In spite of the architectural simplification achieved thanks to the container technology, the amount of technical obstacles can still be cut down with the help of the *cloud* tools and methodologies later described in Section 6. Besides, although the technical requirements of the Docker daemon are very humble, some hardware configurations are too old to efficiently run this technology. Anyway, its mastery is still very motivating for engineering students.

## 5 Simulation of a modern digital hospital network

Once an X-ray image has been generated (either real or simulated), it should be stored according to industry specifications. Needless to say, the standard that governs the interchange of radiological data is DICOM [51]. This protocol is mostly used for storing and transmitting medical images between scanners, modalities, servers, clients, workstations, printers, network hardware, and picture archiving and communication systems (PACS).

In our approach, we encourage students to preserve the obtained X-ray images with the tools and methods described in Section 3.2.5 and Section 3.3 in a real PACS service. However, previously to carry out this request, each image should be packed as a DICOM object. In order to do so, students can use the DCMTK toolkit [52]. This open-source C++ DICOM implementation is offered by the Institute for Information Technology (OFFIS), who allows the download of precompiled binaries for the most common platforms. Among these binaries, it is possible to find `img2dcm` which allows the encapsulation of image files (such as the ones previously obtained through simulations) as DICOM archives:

```
img2dcm image.jpg image.dcm –vlp –k "PatientName=John^Doe"
```

This command also allows the tailoring of the final file with specific DICOM attributes [53], such as the name of the patient. The resulting file can be sent to a reachable PACS with the `storescu` command (also available as part of the DCMTK distribution):

```
storescu –aec AETITLE SERVER PORT image.dcm
```

where: `AETITLE` is the name of the called *Application Entity* (an unique identification name within a DICOM network), `SERVER` is DNS name of the machine hosting the PACS service and `PORT` is the TCP port in which the PACS service is running in the host `SERVER`. Finally, it is necessary to implement the PACS service. There exist plenty of free and open-source solutions that could be used. However, in order to keep this part of the laboratory as simple as possible, we recommend the use of the `dcmqridx` and `dcmqrscp` commands, also available as part of the DCMTK utility tools. The first one builds a simple database index in which information about hosted images will be stored. The second starts the PACS server itself:

```
dcmqridx /root/pacs /root/pacs/*.dcm
dcmqrscp –v –c /root/dcmqrscp.cfg
```

† https://hub.docker.com/u/pammacdotnet

The program `dcmqrscp` needs an external configuration file (`dcmqrscp.cfg`). An example of the contents of this file can be read in Listing 5.

**Listing 5** Minimal PACS configuration for the `dcmqrscp` command (part of the set of utilities available as part of the DCMTK project). This is the simplest configuration possible consisting just in the TCP port to be used, the AETITLE (`PACS`), the folder containing the image database (`/root/pacs`) and allowing connections from any network peer.

```
NetworkTCPPort = 11112
AETable BEGIN
PACS /root/pacs RW (200, 1024mb) ANY
AETable END
```

This simple PACS service can also run containerized with Docker and can be even be deployed in free cloud systems such as the one reviewed in Section 6. It is also available in the Docker Hub† introduced in Section 4.

## 6 Collaborative and cloud-based virtual lab environments

As an extension to the remote virtual labs we have cloud-based ones. They are grounded on the outsourcing of storage and processing capacity to remote and anonymous servers. V-Lab [54] is a clear example that supports both teaching and grading capabilities. As [55] points out, cloud-based labs *decrease the cost of electronic components and the sizes of digital devices and enable the emergence of other models of exploitation*. Cloud-based labs are focused on teaching a diversity of topics, mostly related to computer components that benefit from this outsourced strategy: networks [56], security [57], and raw computing [58].

In recent years, they have evolved to the concept of *Laboratory as a Service* or LaaS, where modular remote laboratories can be constructed, escalated and shared based on the daily needs in the classroom [59]. LaaS make use of a type of virtualization of classic hardware and software components like, i.e. graphics, sound, processors, hard drives, etc. Some of them use Oracle VirtualBox, VMware, Parallels or similar products. They have also evolved to embrace the *container approach* (tackled in Section 4).

### 6.1 Free cloud computing environments

Play With Docker (PWD†) enables the free building and running of Docker containers in the cloud. Play with Docker makes use of *Docker-in-Docker* (DIND) and its implementation is freely available as open-source, making it possible for an institution to host its own PWD service. However, for the sake of simplicity, the official PWD site is recommended. The Play with Docker classroom brings the teacher labs and tutorials that can help him/her get a quick (and most importantly, free) hands-on experience using Docker.

### 6.2 Shared live coding environments

Collaborative editing has been a staple of web authoring tools for many years, with operational transforms [60] enabling writers and programmers to freely edit documents with

† https://hub.docker.com/r/pammacdotnet/dockerpacs
† http://labs.play-with-docker.com

other authors, in a concurrent manner. Synchronous edition of shared documents is favorable for ensemble live-coding performance, as it offers writers/developers the ability to view and edit the work of their fellow performers in real time [61]. These environments allow developers to share a common workflow into a single host, so that they collaborate in real time. They are usually based on encrypted peer-to-peer connections that guarantee privacy and data accuracy, along with a user-tracking activity report.

There exist several environments that allow these types of collaboration scenarios [62], but perhaps, the most used one nowadays is the Live Share† extension to Visual Studio and Visual Studio Code (a professional closed-source IDE and an open-source code editor from Microsoft, respectively). Figure 6 shows an example of a remote virtual classroom in which the teacher (one of the co-authors of this paper) shared a live G4Py (reviewed in Section 3.1) simulation session. Live Share allows co-edition and co-debugging while users can share audio, servers, terminals, diffs, comments, and more.

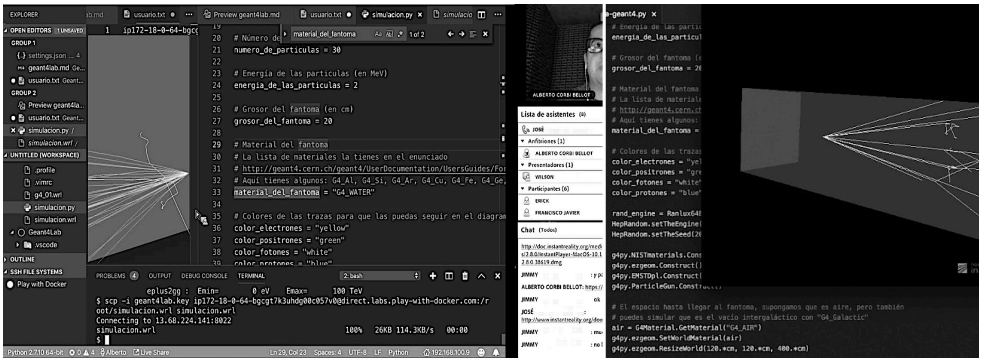

Figure 6: Examples of collaborative sessions with VS Code and its Live Share extension. The Adobe Connect platform is applied for audio and video (other systems can be used).

### 6.3 *Programming languages and environments*

As previously highlighted, a key aspect of our approach is the ability to encourage the learning of modern computer languages, which undoubtedly leverages the motivation of engineering students. The fact of becoming familiar with the underpinnings of X-ray settings allows the interplay with those scripting languages. Any engineer (and even any student in STEAM areas) should master the basic notions of Python, MATLAB, etc. Some reputed experts in pedagogy and learning advise the introduction of computer programming even at earlier stages of the educational process [16]. Python is turning out the most used language in education [63, 64] and digital labs [65, 66]. It is, without the shadow of doubt, a very powerful technology which allows the accomplishment of very different tasks with a few lines of code. It is also widely used in medical physics, imaging and radiology [67]. An example that summons both characteristics is presented in Listing 6.

### 6.4 *Remote notebooks*

Digital notebooks were originally implemented in practice by Wolfram Research in their famous Mathematica software package. However, they were first theoretically introduced as

† https://visualstudio.microsoft.com/services/live-share

**Listing 6** Example Python code for generating the volume information (with a final MetaImage file) from a set of DICOM archives. It uses the SimpleITK open-source library, which is a simplified layer built on top of ITK for rapid prototyping and education purposes.

```
import SimpleITK as sitk
series_IDs = sitk.ImageSeriesReader_GetGDCMSeriesIDs(dirname)
if not series_IDs:
 print('No series in directory ' + '\'' + dirname + '\'')

for s in series_IDs:
 sitk.WriteImage(sitk.ReadImage(sitk.
     ImageSeriesReader_GetGDCMSeriesFileNames(dir, s)), s + '.mhd')
```

literate programming by [68]. A notebook is a document than can store and present static information in various formats (text, images, video, etc.), allow live calculations in a given computing language (originally, the Wolfram Language [69]), return the results of those calculations (numbers, expressions, charts, graphs, tables, video, etc.) and even allow the interaction of the reader through friendly widget- and GUI-based mechanisms. For instance, in the notebook hosted at http://bit.ly/g4pybinder, the students can configure the simulation parameters (type of particles, energy, phantom location, etc.) with visual controls before launching the Geant4-Python simulation.

This same philosophy is the one followed by the *Jupyter* notebooks (formerly known as *IPython* [70]). The Jupyter project allows the creation, delivery and management of interactive homework and lab material in which students can dynamically run code snippets in a web browser, add text and explanations, read exercise statements and *play* with the results. Code snippets can be written in almost any language supported by the Jupyter project through any of the available kernels. As an example of the use of these notebooks (in a radiological context), Figure 7 shows the screenshot of a notebook that allows the student to obtain a bony isosurface from a MetaImage volume (introduced in Section 3.3) and see results in real time. The used Jupyter kernel is, in this case, Python-based. The shown code also makes use of several dependencies such as VTK (from Kitware) or the well-know Numpy.


Figure 7: Left: Some of the cells of a Jupyter notebook for converting a MetaImage volume to a mesh in STL format. The notebook is available at http://bit.ly/volumeisosurface). Right: a screenshot of a live notebook showing a Python+Geant4 medical simulation (hosted at MyBinder in http://bit.ly/g4pybinder).

In theory, a local instance of Jupyter requires the installation of a Python distribution (for both using the notebook server itself and for using the Python kernel with those notebooks

that depend upon it). Besides, a productive Jupyter installation might need several external libraries such as the aforementioned VTK, which do not necessarily come preinstalled. Although there are some nice Python distributions that come with many dependencies pre-packaged (Anaconda† being the most respected one), other solutions allow the remote execution and display of notebooks in free cloud environments (and without the need for any local Python deployment). Perhaps, the most reputed one is Binder (also discussed in [70]). The site MyBinder.org allows running live notebooks from accessible code repositories. For all these reasons, Jupyter is now widely used in education and is becoming a standard for homework delivery [71] in STEAM areas. Together with Binder, it is now possible to finally build reproducible, interactive and sharable environments for science at scale [72].

Furthermore, the LTI Authenticator project† allows identity validation between externally hosted Jupyter instances and any Learning Management System (LMS). The IMS Global Learning Tool Interoperability [73] or LTI standard enables the connection between central campus software (the one in charge of keeping each student grades, etc.) and external content/auditing tools [74], such as those facilitated by the Jupyter notebooks served by foreign institutions, by MOOC providers or transversal cloud services (e.g., MyBinder).

## 7 Geometry of an X-ray camera

As reinforced in Section 1, in a distant education setting, it is very difficult to organize academic visits to real X-ray examination rooms in clinics and hospitals. It is likewise impossible that students have the opportunity to interact (even as mere passive viewers) with real X-ray equipment. Nevertheless, it is still feasible to physically *play* with a similar setup, from home. If we exchange X-rays for plain visible light and real radiographs for the conventional casting of shadows, students can at least gain some insight about the geometry and optics of an X-ray system. Needless to say, this activity is more tangible and *playable* than the previous ones, but it still contains a non-negligible load of programming so that engineering students can keep on enjoying acclaimed computing technologies.

### *7.1 Imitating radiographs with visible light*

As declared in Section 3.2.3, the intrinsic parameters of an X-ray camera vary from examination to examination. With this simple experiment that recreates an X-ray examination with visible light, students can test this fact by themselves. In order to carry it out, learners *fake* a Roentgen pencil beam with a white light source (e.g., the one emitted by a bulb) behind an opaque wrapper with a small hole in it. If the light source is tiny and/or is positioned far enough, we can assume its *point-likeness* and discard the wrapping step. In this way, we can mimic the shape of an X-ray burst that is originated from a visible light pinhole source, which now acts as the anode (i.e., the point C of the optical system).

As with X-rays, the white light will encounter objects on its way and will eventually cast a set of shadows in a wall or planar surface. In an X-ray room, this planar surface would nowadays consist in a FPD that integrates the Roentgen light. This FPD is later physically detached and read in a scanner. In our visible-light proposal, the detection of the X radiation (or its absence) by the FPD and the IP-scanning process is now achieved by taking a simple photograph of the wall/surface in which those shadows are identifiable. In other words, the

† https://www.anaconda.com/distribution
† https://github.com/jupyterhub/ltiauthenticator

role of the detector is now played by an external camera which remains fixed at a specific location during the duration of the experiment and takes a picture (or pictures) of the scene.

In a real X-ray setup, the aforementioned shadows would have been produced with X-ray opaque materials such the commonly used copper fiducials. As with the case of real X-ray reference/calibration frames, we recommend the use of different shapes for each fiducial maker in order to speed up and ease the identification of the corresponding shadows (patterns in the case of radiographs). If the aforementioned objects are placed at known positions relative to each other, it is now possible to perform a camera calibration process, that together with the 2D locations of the casted shadows (available as part of the photograph), reveals the focal distance. This can be accomplished by any of the methods succinctly explained in [75], however, we recommend the DLT algorithm, tackled next.

### 7.2 Calibration algorithm

The Direct Linear Transform (DLT) is a simple algorithm used to obtain the projection matrix (P) given a sufficient set of point correspondences. It was originally devised by Abdel Y. I. Aziz and H. M. Karara [76] and is updated by [77]. DLT estimates the P matrix of an imaging system using a projective transformation and a set of point correspondences. In homogeneous coordinates, each point tuple $\mathbf{q}_i, \mathbf{Q}_i$ presents a ligature which is better written as $\mathbf{q}_i \times \mathrm{P}\mathbf{Q}_i = 0$. From this, a simple linear solution for P can be obtained with 6 correspondences. Figure 8 shows a real example of the proposed setup that the student can replicate at home.

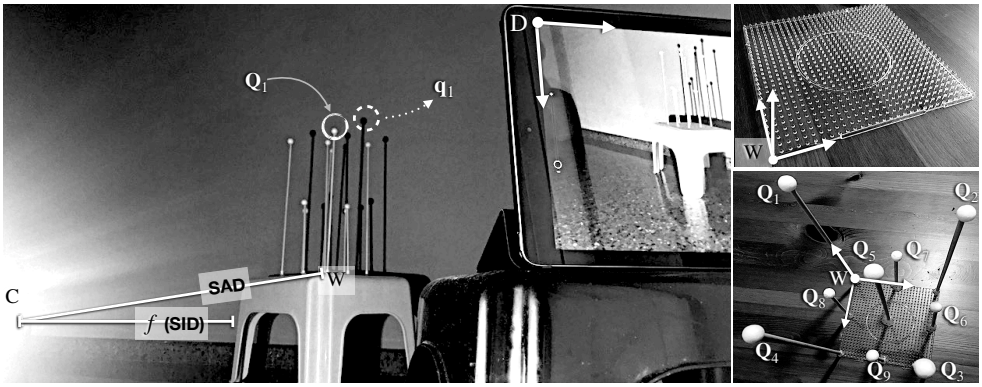


Figure 8: Experiment for studying the geometry of X-rays with *plain* light. Instead of an anode that emits Roentgen radiation, we use a lightbulb that acts as a point-like X-ray source (C). This light casts the spherule-shaped fiducials (e.g., $\mathbf{Q}_1$) as shadows into a wall (e.g., $\mathbf{q}_1$). The 3D positions of the markers have been established deliberately relative to a W reference frame. The role of the detector (D) is now played by an external camera (e.g., the one inside any modern smartphone or tablet PC, as the one in the image). The geometry is equivalent to that shown in Figure 4. On one hand, the SID (or focal length $f$) perpendicularly connects the projection surface (the wall) and the anode C (the bulb). On the other hand, the SAD is the distance from the isocenter or W (defined by the experimenter) to C.

### 7.3 Projection matrix and intrinsic parameters

The *pose* of an object or camera is the combination of position and orientation. It involves the derivation of 5 intrinsic (K sub-matrix) and 6 extrinsic parameters, which can be represented

with a $3 \times 4$ matrix (P, introduced above). Mathematically, P connects 3D points –expressed in W coordinates– to 2D points, using the relation: $\mathbf{q}_i = \mathrm{P} \cdot \mathbf{Q}_i$, where a $\mathbf{q}_i$ is point in the image and $\mathbf{Q}_i$ is a W-related point. The K part of P translates 3D points, expressed in the detector (D) reference frame (Figure 4), to their corresponding image coordinates:

$$\mathrm{K} = \begin{pmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} \lambda_x & 0 & 0 \\ 0 & \lambda_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\lambda} \cdot \begin{pmatrix} f & \sigma & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{6}$$

where $\lambda$ is a *resolution matrix* (*number of pixels per unit of length*), for both $x$ and $y$ axes, and $\alpha_x = f\lambda_x$ and $\alpha_y = f\lambda_y$ represent the focal lengths in pixel units. Similarly, $x_0$ and $y_0$ are the image units counterparts of $c_x$ and $c_y$. The parameter $s$ is the skewness of the camera. If we assume that pixels are square, it is possible to simplify Eq. (6) as:

$$\mathrm{K} = \begin{pmatrix} \alpha & 0 & x_0 \\ 0 & \alpha & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

where we have also deliberately *forced* both focal lengths to be equal to $\alpha$. This assumption makes sense in the field to which our research is intended to contribute, i.e., X-ray diagnostic imaging and item scanning. It can also be applied in the context of the current *visible-light adaptation* here presented. The extrinsic part of P describes a rigid transformation mapping points between W and C frames. This matrix can also be deconstructed in a rotation matrix R (accounting for angles $\theta_x, \theta_y, \theta_z$ ) and the aforestated translation vector $\mathbf{t}$:

$$\overbrace{\left[\mathrm{R} \mid \mathbf{t}\right]}^{\text{extrinsic matrix}} = \overbrace{\underbrace{\left(\begin{array}{c|c} \mathrm{I} & \mathbf{t} \end{array}\right)}_{\text{3D translation}} \cdot \underbrace{\left(\begin{array}{c|c} \mathrm{R} & 0 \\ \hline 0 & 1 \end{array}\right)}_{\text{3D rotation}}}^{\text{extrinsic matrix}} \tag{8}$$

It is common to see a version of Eq. (8) with extra row of $(0,\ 0,\ 0,\ 1)$ added to the bottom. This makes the matrix square, which allows us to further decompose this matrix into the aforementioned rotation and translation parts. Given P, we can decompose it into its intrinsic/extrinsic parts using a QR decomposition. We can do this given the fact that R is orthogonal and K is an upper-triangular matrix:

$$\mathrm{P} = \overbrace{\mathrm{K}}^{\text{intrinsic matrix}} \cdot \overbrace{\left[\mathrm{R} \mid \mathbf{t}\right]}^{\text{extrinsic matrix}} \tag{9}$$

These calculations can be carried out in almost any programming language. In Listing 7, we show a quick example that students can implement with the help of the teacher.

As a side experiment, the student can prove by him/herself how the $f$ (the SID) converges to a more or less constant value as the calculations are repeated with more point-correspondence ($\mathbf{Q}_i \longleftrightarrow \mathbf{q}_i$) combinations.

### *7.4 Estimation of the detector resolution*

Even though radiological technicians have usually access to the detector resolution specified by the manufacturer ($\lambda_o$), this parameter can also be experimentally determined ($\lambda$) from two X-ray images produced at different geometrical configurations [75]. In our case, the student

**Listing 7** MATLAB/GNU Octave code to calibrate a camera from real and image points. The last line returns the focal length ($f$) of the system (also referred as SID, as discussed in Figure 4), which for case of an X-ray system is different each time. This situation can be easily emulated with a visible light setup. The DLT and QR subroutines are usually available as part of the computer vision toolbox of the aforementioned computing environments.

```
real_points = [0 0 10; 10 0 5; 10 10 10; 0 10 15];
points_in_photograph = [205 158 0; 430 230 0; 1810 767 0; 1110 16 0];
P = DLT(points_in_photograph, real_points);
[K, R, t] = QR(p_x_calib);
K = K./K(3,3);
focal_length = K(1,1); % equal to the SID
SAD = norm(t);
```

can carry out a similar test with a *visible-light alternative* and, instead of making use of two radiographs, now two photographs will come into play. Of course, now the detector is the external camera (used to take the photographs of the wall where the shadows are casted), so the obtained resolution should match more or less the factory resolution of this sensor.

If we take a look at Figure 9, we can easily notice that the distances represented by the vector relations $|\mathbf{p}_2 - \mathbf{p}_1|$ and $|\mathbf{t}_2 - \mathbf{t}_1|$ account for the same spatial gap. However, there exists an important difference between the two: the former is expressed in pixel units and the latter is specified in physical units. This fact enables the calculation of $\lambda$ with the expression:

$$\lambda \cdot |\mathbf{t}_2 - \mathbf{t}_1| = |\mathbf{p}_2 - \mathbf{p}_1| \tag{10}$$

where the tuples $\mathbf{p}_2, \mathbf{p}_1$ and $\mathbf{t}_2, \mathbf{t}_1$ represent the anode coordinates in D and W reference frames, for X-ray source locations 1 and 2, respectively. That is, we experimentally resolve how many pixels/meter ($\lambda$) holds the FPD (for a given *stereo snapshot*), which should be *a priori* close to factory specs $\lambda \approx \lambda_o$.

As stated above, the $\mathbf{t}$ vectors have natural units (m, mm, etc.) and the $\mathbf{p}$ vectors have pixel units so that when divided, we obtain a unit for linear resolution (i.e., pixels per meter). Both $\mathbf{p}$ and $\mathbf{t}$ vector pairs are derived with a QR decomposition of $P_1$ and $P_2$ representing the camera calibration at each of the two anode locations/orientations:

$$\mathrm{P} \rightarrow \mathrm{RQ\ decomposition} \rightarrow \underbrace{\begin{bmatrix} \alpha & s & x_0 \\ 0 & \alpha & y_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{p}=(x_0,y_0,\alpha)} \cdot [\mathrm{R} \mid \mathbf{t}] \tag{11}$$

where each vector $\mathbf{p}$ is built exclusively from the same intrinsic parameters already expressed in Eq. (7), and $\mathbf{t}$ is nothing else but the *translation vector*, part of the extrinsic side of the camera matrix equation. This QR decomposition can be carried out with any of the useful programming languages, tools and environments tackled in this text. In Listing 8, we show a basic GNU Octave code to carry out this calculation. A GNU Octave runtime can be easily installed by the students in their own systems or they can use Jupyter-ready cloud services (described in Section 6.4) with GNU Octave kernels. Docker containers in which the aforementioned MATLAB clone has been previously installed can also be used, either locally or also in the cloud (as discussed in Section 6.1). Finally and as it can be seen, the experiment can be carried out at home with relatively easiness and with cheap materials.

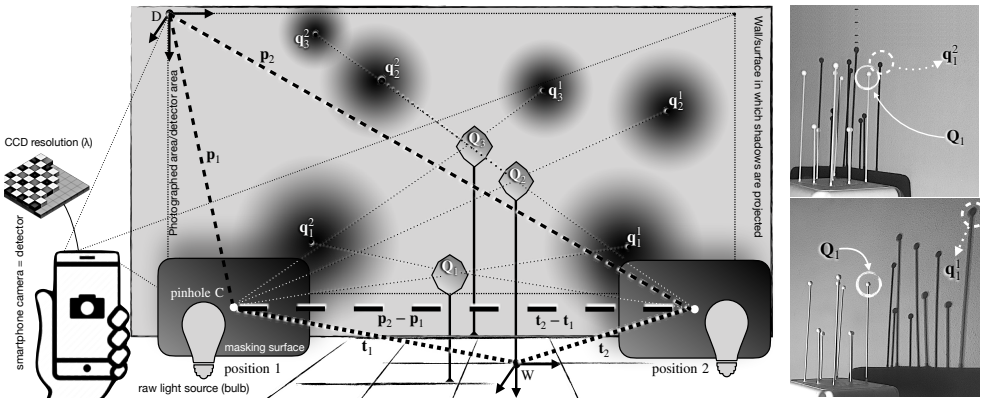


Figure 9: Left: Derivation of detector linear resolution ($\lambda$). The X-ray equipment (the visible-light setup in this specific case) is used at two different locations (1 and 2) and two images (photographs) are generated at each system pose. Taking into account the pair of vectors connecting the anode and the world ($\mathbf{t}_1$ and $\mathbf{t}_2$) and the pair of vectors linking the anode and the detector ($\mathbf{p}_1$ and $\mathbf{p}_2$), it is possible to derive $\lambda$. Both photographs should be taken from the same position. Example of a real implementation of the schematic shown in Figure 9. Right: Two images (1-top and 2-bottom) are produced by the *detector* (a consumer tablet computer with webcam) at two different positions of the *anode* (i.e., white light bulb in our case). The images also show the 2D location of the projection of one of the fiducials: $\mathbf{q}_1^1$ at pixel $(840, 30)$ and $\mathbf{q}_1^2$ at pixel $(610, 60)$. As it can be seen, if all the fiducials have the same shape, it is some difficult to identify their shadows without uncertainty.

**Listing 8** Octave code to calculate the detector resolution ($\lambda$/`lambda`) in an X-ray setting (or the *visible light imitating scenario*). The vectors `p1` and `p2` are derived from the intrinsics contained in `K1` and `K2`. The vectors `t1` and `t2` are obtained as part of the calibration phase.

```
p1 = [K1(1,3) K1(2,3) K1(1,1)];
p2 = [K2(1,3) K2(2,3) K2(1,1)];
lambda = abs(p2 - p1)/abs(t2 - t1);
```

## 8 Academic experience and results

The tools, methods, experiments and activities presented in this work have been offered to online engineering students and carried out as part of the subject *physics for computer scientists* that is held in the first semester at Universidad Internacional de La Rioja (UNIR). In addition to the previously presented results, setups, simulations, and experiments, Figure 10 and Figure 11 show some more examples of the source material (volumes, CT slices, isosurfaces, etc.) and the results that can be obtained within the context of the proposed exercises (mainly, computer generated radiographs). The virtual lab presented in this text has enabled the learning of complex physics concepts while keeping student motivation high and while encouraging future engineers and technology enthusiasts to master the most recent computing standards and tools. It has also emphasized the awareness and respect about health, medicine, healthcare technology and the discipline of radiology.

As just stated, a set of very similar activities to the ones here discussed has been proposed to the students along several semesters in the aforementioned academic institution, being 9.1

the average mark (in a 0 to 10 range). Student satisfaction with the subject, materials and methodology are also one of the highest of this computer science degree.

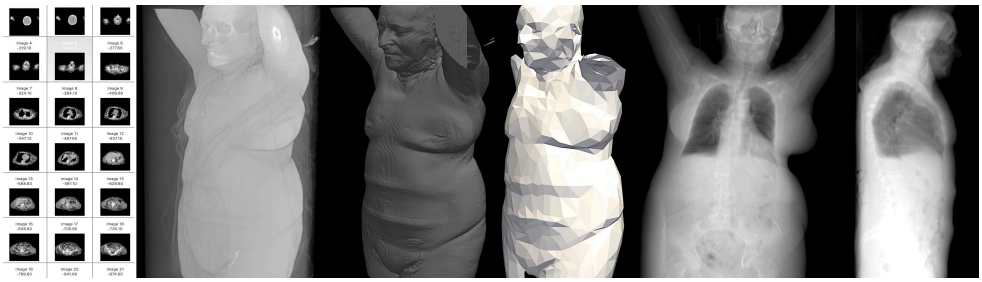

Figure 10: Example of a CT study that has been converted into a MetaImage volume in order to generate a digitally reconstructed radiograph from two perspectives (two rightmost images). The two center images show the skin value isosurface of that volume.

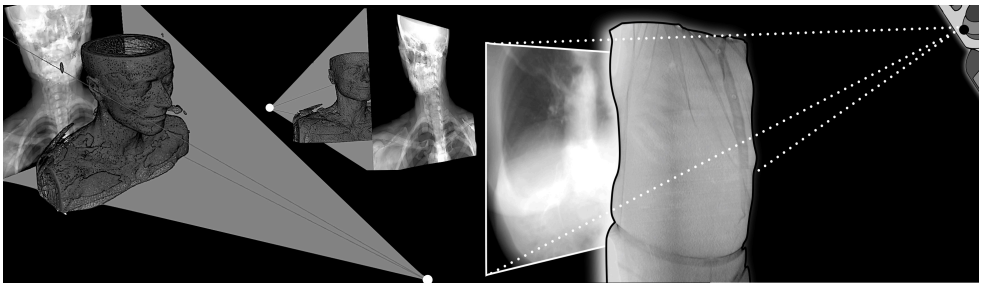

Figure 11: Several DDRs obtained from virtual volumes (corresponding to skin-density isosurfaces). The 3D compositions have been done in gVirtualXRay and Kitware's Paraview, respectively. Both tools (discussed in this text) allow students to see (and more importantly, *intuitively grasp*) the 3D scene of the X-ray examination as if they were technicians.

## 9 Conclusions

We have presented the underpinnings of a remote virtual lab for learning the basics of plain X-ray imaging. The experiments and activities have been specifically designed with a steady degree of difficulty in mind and to be executed mainly in distant learning higher education scenarios. The laboratory is addressed to engineering college students and it enables the practice with modern technologies closely related to the process of generating primary care radiographs. These technologies comprise novel computer languages, virtualization efforts and genuine applications very related to the realm of engineering disciplines. All the proposed tools/experiments can be effortlessly executed without the need for complex software deployments. In fact, almost all of them can directly run in web browsers and free cloud services. In spite of that, the technical/low level background of some tools has also been briefly described in order to foster the motivation of the aforestated students. The paper also tries to account for the freedom of choice that online engineering students have at their disposal in order to learn a basic science through joyful computing environments applied to medical physics. These environments range from their most pro to their out-of-the-box

versions. In addition to the software-based solutions, a more practical task that mimics the geometry in X-ray settings has been proposed. The described lab has been carried out along several semesters in an online academic institution with acclaimed success among students.

## Acknowledgments

This research is partially funded by Universidad Internacional de la Rioja (UNIR, http://transfer.unir.net) through the Research Institute for Innovation & Technology in Education (UNIR iTED, http://ited.unir.net).

**ORCID IDs** A. Corbi 0000-0002-7282-4557, D. Burgos 0000-0003-0498-1101, F. Vidal 0000-0002-2768-4524, A. Albiol 0000-0002-1970-3289, F. Albiol 0000-0002-3824-2246

## References

[1] Louis Deslauriers, Ellen Schelew, and Carl Wieman. Improved learning in a large-enrollment physics class. *Science*, 332(6031):862–864, 2011.

[2] Barton K Pursel, Liang Zhang, Kathryn W Jablokow, GW Choi, and Darrell Velegol. Understanding mooc students: motivations and behaviours indicative of mooc completion. *Journal of Computer Assisted Learning*, 32(3):202–217, 2016.

[3] Balakrishnan Balamuralithara and Peter Charles Woods. Virtual laboratories in engineering education: the simulation lab and remote lab. *Computer Applications in Engineering Education*, 17(1):108–118, 2009.

[4] J Sánchez, F Morilla, S Dormido, J Aranda, and P Ruipérez. Virtual and remote control labs using Java: a qualitative approach. *IEEE Control systems magazine*, 22(2):8–20, 2002.

[5] Ruben Heradio, Luis de la Torre, Daniel Galan, Francisco Javier Cabrerizo, Enrique Herrera-Viedma, and Sebastian Dormido. Virtual and remote labs in education: a bibliometric analysis. *Computers & Education*, 98:14–38, 2016.

[6] Heinz-Dietrich Wuttke, Karsten Henke, and Nadine Ludwig. Remote labs versus virtual labs for teaching digital system design. In *International Conference on Computer Systems and Technologies-CompSysTech*, pages 2–1, 2005.

[7] Ranjan Bose. Virtual labs project: a paradigm shift in Internet-based remote experimentation. *IEEE access*, 1:718–725, 2013.

[8] Elio Sancristobal, Manuel Castro, Sergio Martin, Mohamed Tawkif, Alberto Pesquera, Rosario Gil, Gabriel Díaz, and Juan Peire. Remote labs as learning services in the educational arena. In *Global Engineering Education Conference (EDUCON)*, pages 1189–1194. IEEE, 2011.

[9] Pablo Orduña, Luis Rodriguez-Gil, Javier Garcia-Zubia, Ignacio Angulo, Unai Hernandez, and Esteban Azcuenaga. Labsland: a sharing economy platform to promote educational remote laboratories maintainability, sustainability and adoption. In *Frontiers in Education Conference (FIE)*, pages 1–6. IEEE, 2016.

[10] Yves De Deene. Optical CT scanning for experimental demonstration of medical X-ray CT and SPECT. *European Journal of Physics*, 40(2):024001, 2019.

[11] Alberto Albiol, Alberto Corbi, and Daniel Burgos. Design of a remote signal processing student lab. *IEEE Access*, 5:16068–16076, 2017.

[12] Susanna Guatelli, Catherine Layton, Dean Cutajar, and Anatoly B Rosenfeld. The teaching/research nexus and internationalisation: An action research project in radiation physics. *Journal of University Teaching and Learning Practice*, 7(2):5, 2010.

[13] Christopher M Poole, Iwan Cornelius, Jamie V Trapp, and Christian M Langton. Technical Note: Radiotherapy dose calculations using Geant4 and the Amazon Elastic Compute Cloud. *arXiv e-prints*, 2011.

[14] Gary White. Cognitive characteristics for learning C++. *Journal of Computer Information Systems*, 42(3):51–55, 2002.

[15] K Murakami and H Yoshida. A Geant4-Python interface: development and its applications. In *Nuclear Science Symposium Conference*, volume 1, pages 98–100. IEEE, 2006.

[16] Linda Grandell, Mia Peltomäki, Ralph-Johan Back, and Tapio Salakoski. Why complicate things? Introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*, pages 71–80. Australian Computer Society, Inc., 2006.

[17] K Murakami, K Amako, J Jacquemier, M Maire, and H Yoshida. Geant4 simulation for education in medical application. In *2008 IEEE Nuclear Science Symposium Conference Record*, pages 3169–3171. IEEE, 2008.

[18] Yuichiro Anzai and Herbert A Simon. The theory of learning by doing. *Psychological review*, 86(2):124, 1979.

[19] Roger C Schank, Tamara R Berman, and Kimberli A Macpherson. Learning by doing. *Instructional-design theories and models: A new paradigm of instructional theory*, 2(2):161–181, 1999.

[20] Brian R Kent. *3D scientific visualization with Blender*. Morgan & Claypool, 2014.

[21] Tom McCauley. Iop: A browser-based event display for the cms experiment at the lhc using webgl. In *J. Phys.: Conf. Ser.*, volume 898, page 072030, 2017.

[22] Kenton McHenry and Peter Bajcsy. An overview of 3D data content, file formats and viewers. *National Center for Supercomputing Applications*, 1205:22, 2008.

[23] P.-F. Villard, F. P. Vidal, C. Hunt, F. Bello, N. W. John, S. Johnson, and D. A. Gould. A prototype percutaneous transhepatic cholangiography training simulator with real-time breathing motion. *International Journal of Computer Assisted Radiology and Surgery*, 4(6):571–578, 2009.

[24] Aaron Sújar, Andreas Meuleman, Pierre-Frederic Villard, Marcos García, and Franck P. Vidal. gVirtualXRay: Virtual X-Ray Imaging Library on GPU. In Tao Ruan Wan and Franck Vidal, editors, *Computer Graphics and Visual Computing (CGVC)*. The Eurographics Association, 2017.

[25] Z. Zuo, W. Y. Qian, X. Liao, and P. Heng. Position based catheterization and angiography simulation. In *2018 IEEE 6th International Conference on Serious Games and Applications for Health (SeGAH)*, pages 1–7, May 2018.

[26] Franck P. Vidal and Pierre-Frédéric Villard. Simulated Motion Artefact in Computed Tomography. In *Eurographics Workshop on Visual Computing for Biology and Medicine*. Eurographics Association, 2015.

[27] F. P. Vidal. gVirtualXRay - Fast X-ray simulation on GPU. In *Workshop on Image-Based Finite Element Method for Industry 2018 (IBFEM-4i 2018)*, Swansea, UK, September 2018. Invited talk.

[28] F.P. Vidal, J.M. Létang, G. Peix, and P. Cloetens. Investigation of artefact sources in synchrotron microtomography via virtual X-ray imaging. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 234(3):333 – 348, 2005.

[29] Hafeez Anwar, Irfanud Din, and Kang Park. Projector calibration for 3d scanning using virtual target images. *International Journal of Precision Engineering and Manufacturing*, 13(1):125–131, 2012.

[30] F. P. Vidal. gVirtualXRay: Virtual X-Ray Imaging Library on GPU, 2019.

[31] N. Freud, P. Duvauchelle, J. M. Létang, and D. Babot. Fast and robust ray casting algorithms for virtual x-ray imaging. *Nucl Instrum Methods Phys Res B*, 248(1):175–180, 2006.

[32] Franck P. Vidal, Manuel Garnier, Nicolas Freud, Jean Michel Létang, and Nigel W. John. Simulation of X-ray Attenuation on the GPU. In Wen Tang and John Collomosse, editors, *Theory and Practice of Computer Graphics*. Eurographics Association, 2009.

[33] Simon Green. The OpenGL framebuffer object extension. In *Game developers conference*, 2005.

[34] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[35] Franck P. Vidal and Pierre-Frédéric Villard. Development and validation of real-time simulation of x-ray imaging with respiratory motion. *Computerized Medical Imaging and Graphics*, 49:1 – 15, 2016.

[36] F. P. Vidal, M. Garnier, N. Freud, J.M. Létang, and N.W. John. Accelerated Deterministic Simulation of X-ray Attenuation Using Graphics Hardware. In Anders Hast and Ivan Viola, editors, *Eurographics 2010 - Posters*. The Eurographics Association, 2010.

[37] Alfio Quarteroni Fausto Saleri and Paola Gervasio. Scientific Computing with MATLAB and Octave. *AMC*, 10:12, 2006.

[38] David M Beazley. Automated scientific software scripting with swig. *Future Generation Computer Systems*, 19(5):599–609, 2003.

[39] Will Schroeder, Lydia Ng, Josh Cates, et al. The ITK software guide. *The Insight Consortium*, 2003.

[40] James Shackleford, Nadya Shusharina, Joost Verberg, Guy Warmerdam, Brian Winey, Markus Neuner, Philipp Steininger, Amelia Arbisser, Polina Golland, Yifei Lou, Chiara Paganelli, Marta Peroni, Marco Riboldi, Guido Baroni, Paolo Zaffino, MariaFrancesca Spadea, Aditya Apte, Ziad Saleh, JosephO Deasy, Shinichro Mori, Nagarajan Kandasamy, and GregoryC Sharp. Plastimatch 1.6 - current capabilities and future directions. *Int Conf Med Image Comput Comput Assist Interv*, 15, 2012.

[41] Paolo Zaffino, Patrik Raudaschl, Karl Fritscher, Gregory C Sharp, and Maria Francesca Spadea. Plastimatch MABS, an open source tool for automatic image segmentation. *Medical Physics*, 43(9):5155–5160, 2016.

[42] P Zaffino, P Raudaschl, K Fritscher, M Spadea, and G Sharp. Validation of Plastimatch MABS, a Tool for Automatic Image Segmentation. *Medical Physics*, 43(6):3658–3658, 2016.

[43] R L Siddon. Prism representation: a 3D ray-tracing algorithm for radiotherapy applications. *Physics in Medicine and Biology*, 30(8):817, 1985.

[44] Xun Jia and Steve B Jiang. *Graphics Processing Unit-Based High Performance Computing in Radiation Therapy*. CRC Press, 2015.

[45] Susanta Nanda Tzi-cker Chiueh and Stony Brook. A survey on virtualization technologies. *Rpe Report*, 142, 2005.

[46] Michael Still. *The definitive guide to ImageMagick*. Apress, 2006.

[47] Daniel Burgos and Alberto Corbi. Transgenic learning for STEAM subjects and virtual containers for OER. *Distance Education*, 39(1):4–18, 2018.

[48] Alberto Corbi and Daniel Burgos. Open Distribution of Virtual Containers as a Key Framework for Open Educational Resources and STEAM Subjects. *Electronic Journal of e-Learning*, 15(2):126–136, 2017.

[49] Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[50] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[51] Peter Mildenberger, Marco Eichelberg, and Eric Martin. Introduction to the DICOM standard. *European Radiology*, 12(4):920–927, 2002.

[52] Marco Eichelberg, Joerg Riesmeier, Thomas Wilkens, Andrew J Hewett, Andreas Barth, and Peter Jensch. Ten years of medical imaging standardization and prototypical implementation: the DICOM standard and the OFFIS DICOM toolkit (DCMTK). In *Medical Imaging 2004: PACS and Imaging Informatics*, volume 5371, pages 57–69. International Society for Optics and Photonics, 2004.

[53] NEMA. Dicom part 6: Data dictionary. Technical report, DICOM, 2011.

[54] Le Xu, Dijiang Huang, and Wei-Tek Tsai. V-lab: a cloud-based virtual laboratory platform for hands-on networking courses. In *Conference on Innovation and technology in computer science education*, pages 256–261. ACM, 2012.

[55] Hamadou Saliah-Hassane, Maarouf Saad, Willie K Ofosu, Djibo Karimou, and Hassane Mayaki Alzouma. Lab@home: Remote laboratory evolution in the cloud computing era. American Society for Engineering Education, 2011.

[56] Razvan I Dinita, George Wilson, Adrian Winckles, Marcian Cirstea, and Aled Jones. A cloud-based virtual computing laboratory for teaching computer networks. In *International Conference on Optimization of Electrical and Electronic Equipment (OPTIM)*, pages 1314–1318. IEEE, 2012.

[57] Le Xu, Dijiang Huang, and Wei-Tek Tsai. Cloud-based virtual laboratory for network security education. *IEEE Transactions on Education*, 57(3):145–150, 2013.

[58] Alexey Dukhanov, Maria Karpova, and Klavdiya Bochenina. Design virtual learning labs for courses in computational science with use of cloud computing technologies. *Procedia Computer Science*, 29:2472–2482, 2014.

[59] Mohamed Tawfik, Christophe Salzmann, Denis Gillet, David Lowe, Hamadou Saliah-Hassane, Elio Sancristobal, and Manuel Castro. Laboratory as a Service (LaaS): A model for developing and implementing remote laboratories as modular components. In *International Conference on Remote Engineering and Virtual Instrumentation (REV)*, pages 11–20. IEEE, 2014.

[60] Clarence A Ellis and Simon J Gibbs. Concurrency control in groupware systems. In *Sigmod Record*, volume 18, pages 399–407. ACM, 1989.

[61] Sang Won Lee and Georg Essl. Communication, control, and state sharing in networked collaborative live coding. *Ann Arbor*, 1001:48109–2121, 2014.

[62] Charlie Roberts, Karl Yerkes, Danny Bazo, Matthew Wright, and JoAnn Kuchera-Morin. Sharing time and code in a browser-based live coding environment. In *International Conference on Live Coding (ICSRiM)*, pages 179–185, 2015.

[63] Frank Stajano. Python in education: Raising a generation of native speakers. In *Proceedings of 8 th International Python Conference*, pages 2000–01, 2000.

[64] Pai H Chou. Algorithm education in Python. *Proceedings of Python*, 10:2, 2002.

[65] Atanas Radenski. Python first: A lab-based digital introduction to computer science. *ACM SIGCSE Bulletin*, 38(3):197–201, 2006.

[66] Claudiu Chiculita and Laurentiu Frangu. A web based remote control laboratory. In *6th Multiconference on Systemic, Cybernetics and Informatics, Orlando, Florida, Etats-Unis*, pages 14–18, 2002.

[67] Ravishankar Chityala and Sridevi Pudipeddi. *Image processing and acquisition using Python*. Chapman and Hall/CRC, 2014.

[68] D E Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[69] George Mias. A Wolfram Language Primer for Bioinformaticians. In *Mathematica for Bioinformatics*, pages 7–65. Springer, 2018.

[70] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter Notebooks: a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[71] Alberto Corbi and Daniel Burgos. OERaaS: Open educational resources as a service with virtual containers. *IEEE Latin America Transactions*, 14(6):2927–2933, 2016.

[72] Jupyter et al. Binder 2.0-reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th Python in Science Conference*, 2018.

[73] Charles Severance, Ted Hanss, and Josepth Hardin. Ims learning tools interoperability: Enabling a mash-up approach to teaching and learning tools. *Technology, Instruction, Cognition and Learning*, 7(3-4):245–262, 2010.

[74] Alberto Corbi and Daniel Burgos. Review of current student-monitoring techniques used in elearning-focused recommender systems and learning analytics. *IJIMAI*, 2(7):44–52, 2014.

[75] Francisco Albiol, Alberto Corbi, and Alberto Albiol. Evaluation of modern camera calibration techniques for conventional diagnostic X-ray imaging settings. *Radiological Physics and Technology*, 10(1):68–81, 2017.

[76] Abdel Y. I. Aziz and H. M. Karara. Direct linear transformation into object space coordinates in close-range photogrammetry. In *Proc. of the Symposium on Close-Range Photogrammetry*, pages 1–18, Urbana, Illinois, 1971.

[77] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.